UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/816,248	03/31/2004	Youfeng Wu	42P18509	9130

59796 7590 07/03/2007

INTEL CORPORATION
c/o INTELLEVATE, LLC
P.O. BOX 52050
MINNEAPOLIS, MN 55402

EXAMINER	
WANG, BEN C	
ART UNIT	**PAPER NUMBER**
2192	
MAIL DATE	**DELIVERY MODE**
07/03/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/816,248	WU ET AL.	
	Examiner	Art Unit	
	Ben C. Wang	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *31 March 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-38* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-12 and 15-38* is/are rejected.

7)☐ Claim(s) *13 and 14* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *4/20/2005*.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

1. Claims 1-38 are pending in this application and presented for examination.

Specification Objections

2. The specification is objected to because the following informalities:

- "Sun Microsystems", cited in [0007], Line 1, is a registered trademark
- "The node <u>305</u> passes the software version information", cited in [0080], Line 3, should be corrected as "The node <u>302</u> passes the software version information"

Appropriate correction is required (See MPEP § 608.01(b))

Claim Rejections – 35 USC § 102(e)

The following is quotation of 35 U.S.C. 102(e) which form the basis for all obviousness rejections set forth in this office action:

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

3. Claims 1-2, 14-15, 23, and 31 are rejected under 35 U.S.C. 102(e) as being anticipated by Roediger et al. (Pat. No. US 6,938,249 B2) (hereinafter 'Roediger')

4. **As to claim 1**, Roediger discloses a method, comprising collecting a loop trip count continuously during runtime of a region of code being executed that contains a

loop (Fig. 2, step 230 – user runs instrumented program on sample inputs to gather profile data; Col. 2, Lines 4-8 – a profile-based loop optimizer generates an execution frequency table for each loop that gives more detailed profile data that allows making a more intelligent decision regarding if and how to optimize each loop in the computer program; Col. 7, Lines 13-19 – new instrumentation code is generated for each loop that collects profile data in an execution frequency table that corresponds to the loop; the execution frequency table gives enhanced information that allows a more intelligent choice of whether to peel or unroll a loop according to a dominant mode, if present, in the execution frequency table); categorizing the trip count to identify one or more code modification techniques applicable to the loop (Col. 2, Lines 12-15 – the execution frequency table is used to determine whether there is one dominant mode that appears in the profile data, and if so, optimizes the loop according to the dominant mode; Fig. 11, steps 1120, 1130); and dynamically applying the one or more applicable code modification techniques to alter the code that relates to the loop (Fig. 2, step 240 – back-end compiler retranslates IR (intermediate representation) code into machine code, applying profile data to enhance optimization; Col. 2, Lines 15-19 – the optimizer may perform optimizations by peeling a loop, by unrolling a loop, and by performing both peeling and unrolling on a loop according to the profile data in the execution frequency table for the loop; Fig. 10, step of 1050 – continue with compilation, applying loop optimizations according to values recorded in frequency execution tables; Col. 8, Lines 10-13 – one specific method may be performed during step 1050 of Fig. 10 to optimize one or more loops according to the profile data stored in the execution

frequency tables; Fig. 11, steps of 1140, 1150, 1160, 1132, 1142, 1152, 1162; Col. 8, Lines 10-53).

5. **As to claim 15**, Roediger discloses a method comprising, repeatedly categorizing a loop trip count that is evaluated continuously during runtime (Fig. 2, step 230 – user runs instrumented program on sample inputs to gather profile data; Col. 2, Lines 4-8 – a profile-based loop optimizer generates an execution frequency table for each loop that gives more detailed profile data that allows making a more intelligent decision regarding if and how to optimize each loop in the computer program; Col. 7, Lines 13-19 – new instrumentation code is generated for each loop that collects profile data in an execution frequency table that corresponds to the loop; the execution frequency table gives enhanced information that allows a more intelligent choice of whether to peel or unroll a loop according to a dominant mode, if present, in the execution frequency table); determining after each categorization whether to apply one or more modification techniques to the loop if the categorization meets one or more criteria (Col. 2, Lines 12-15 – the execution frequency table is used to determine whether there is one dominant mode that appears in the profile data, and if so, optimizes the loop according to the dominant mode; Fig. 11, steps 1120, 1130); and dynamically applying the one or more applicable modification techniques to the loop based on the one or more criteria that are met (Fig. 2, step 240 – back-end compiler retranslates IR (intermediate representation) code into machine code, applying profile data to enhance optimization; Col. 2, Lines 15-19 – the optimizer may perform

optimizations by peeling a loop, by unrolling a loop, and by performing both peeling and unrolling on a loop according to the profile data in the execution frequency table for the loop; Fig. 10, step of 1050 – continue with compilation, applying loop optimizations according to values recorded in frequency execution tables; Col. 8, Lines 10-13 – one specific method may be performed during step 1050 of Fig. 10 to optimize one or more loops according to the profile data stored in the execution frequency tables; Fig. 11, steps of 1140, 1150, 1160, 1132, 1142, 1152, 1162; Col. 8, Lines 10-53).

6. **As to claim 23**, Roediger discloses a machine readable medium having embodied thereon instructions, which when executed by a machine, causes the machine to perform a method comprising collecting a loop trip count continuously during runtime of a region of code being executed that contains a loop (Fig. 2, step 230 – user runs instrumented program on sample inputs to gather profile data; Col. 2, Lines 4-8 – a profile-based loop optimizer generates an execution frequency table for each loop that gives more detailed profile data that allows making a more intelligent decision regarding if and how to optimize each loop in the computer program; Col. 7, Lines 13-19 – new instrumentation code is generated for each loop that collects profile data in an execution frequency table that corresponds to the loop; the execution frequency table gives enhanced information that allows a more intelligent choice of whether to peel or unroll a loop according to a dominant mode, if present, in the execution frequency table); categorizing the trip count to identify one or more code modification techniques applicable to the loop (Col. 2, Lines 12-15 – the execution frequency table is used to

determine whether there is one dominant mode that appears in the profile data, and if so, optimizes the loop according to the dominant mode; Fig. 11, steps 1120, 1130); and dynamically applying the one or more applicable code modification techniques to alter the code that relates to the loop (Fig. 2, step 240 – back-end compiler retranslates IR (intermediate representation) code into machine code, applying profile data to enhance optimization; Col. 2, Lines 15-19 – the optimizer may perform optimizations by peeling a loop, by unrolling a loop, and by performing both peeling and unrolling on a loop according to the profile data in the execution frequency table for the loop; Fig. 10, step of 1050 – continue with compilation, applying loop optimizations according to values recorded in frequency execution tables; Col. 8, Lines 10-13 – one specific method may be performed during step 1050 of Fig. 10 to optimize one or more loops according to the profile data stored in the execution frequency tables; Fig. 11, steps of 1140, 1150, 1160, 1132, 1142, 1152, 1162; Col. 8, Lines 10-53).

7. **As to claim 31**, Roediger discloses a system, comprising: a bus (Fig. 21, element of 2160 – Bus); a processor coupled to the bus (Fig. 21, element of 2110 – Processor); a network interface card coupled to the bus (Fig. 21, element of 2150 – Network I/F); and memory coupled to the processor (Fig. 21, element of 2120 – Main Memory), the memory adapted for storing instructions (Fig. 21, elements of 2127 – Compiler, 2128 – Loop Optimizer) (Col. 11, Lines 55-59 – computer system comprises a processor , a main memory,.., and a network interface; Col. 12, Lines 8-11 – compiler includes a loop optimizer that may optimize loops in the intermediate representation

according to profile data stored in the execution frequency table; Col. 13, Lines 26-29 – some of these resources are processor, main memory, ..., network interface, and system bus;), which upon execution by the processor collects a loop trip count continuously during runtime of a region of code being executed that contains a loop (Fig. 2, step 230 – user runs instrumented program on sample inputs to gather profile data; Col. 2, Lines 4-8 – a profile-based loop optimizer generates an execution frequency table for each loop that gives more detailed profile data that allows making a more intelligent decision regarding if and how to optimize each loop in the computer program; Col. 7, Lines 13-19 – new instrumentation code is generated for each loop that collects profile data in an execution frequency table that corresponds to the loop; the execution frequency table gives enhanced information that allows a more intelligent choice of whether to peel or unroll a loop according to a dominant mode, if present, in the execution frequency table), categorizes the trip count to identify one or more code modification techniques applicable to the loop (Col. 2, Lines 12-15 – the execution frequency table is used to determine whether there is one dominant mode that appears in the profile data, and if so, optimizes the loop according to the dominant mode; Fig. 11, steps 1120, 1130), and dynamically applies the one or more applicable code modification techniques to alter the code that relates to the loop (Fig. 2, step 240 – back-end compiler retranslates IR (intermediate representation) code into machine code, applying profile data to enhance optimization; Col. 2, Lines 15-19 – the optimizer may perform optimizations by peeling a loop, by unrolling a loop, and by performing both peeling and unrolling on a loop according to the profile data in the execution

frequency table for the loop; Fig. 10, step of 1050 – continue with compilation, applying loop optimizations according to values recorded in frequency execution tables; Col. 8, Lines 10-13 – one specific method may be performed during step 1050 of Fig. 10 to optimize one or more loops according to the profile data stored in the execution frequency tables; Fig. 11, steps of 1140, 1150, 1160, 1132, 1142, 1152, 1162; Col. 8, Lines 10-53).

8. **As to claim 2** (incorporating the rejection in claim 1), Roediger discloses the method wherein collecting a loop trip count further comprises: collecting a trip count for the loop each time the loop is entered; and calculating an average trip count using a sequential plurality of collected trip counts over an interval of time (Fig. 5; Col. 5, Line 46 through Col. 6, Line 7 – a known method in the prior art for determining the average number of executions per loop entry is shown as method 500 in Fig. 5; the total number of loop execution is then divided by the total number of loop entries to derive the average number of loop executions per loop entry).

9. **As to claim 14** (incorporating the rejection in claim 2), the interval of time is equal to one second.

However, it is well known in the art of arbitrarily selecting a specific interval of time in order to obtain the benefits know in the art.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

10. Claims 3, 5-8, 12-13, 16-22, 24-25, 27-30, 32-33, and 35-38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Roediger in view of Chen et al., (*Dynamic Trace Selection Using Performance Monitoring Hardware Sampling, March 2003, IEEE*) (hereinafter 'Chen')

11. **As to claim 3** (incorporating the rejection in claim 2), Roediger does not explicitly disclose the method further comprising executing the region of code for an introductory profiling phase time interval to establish an initial average trip count value.

However, in an analogous art of dynamic trace selection using performance monitoring hardware sampling, Chen discloses the method further comprising executing the region of code for an introductory profiling phase time interval to establish an initial average trip count value (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the beginning of program; for

simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Chen into the Roediger's system to further provide executing the region of code for an introductory profiling phase time interval to establish an initial average trip count value in Roediger system.

The motivation is that it would further enhance the Roediger's system by taking, advancing and/or incorporating Chen's system which offers significant advantages for capturing 58% of execution time across various SPEC2000 integer benchmarks using our profile and patching techniques on a relatively small number of frequently executed execution paths as once suggested by Chen (e.g., Abstract, 3rd Par.).

12. **As to claim 5** (incorporating the rejection in claim 3), Roediger discloses the method wherein categorizing the trip count further comprises: determining a low trip count threshold value and a high trip count threshold value; classifying the trip count as being in a first condition if the trip count is equal to or below the low trip count threshold value; classifying the trip count as being in a second condition if the trip count is above the low trip count threshold value and below the high trip count threshold value; and classifying the trip count as being in a third condition if the trip count is equal to or above the high trip count threshold value (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

13. **As to claim 6** (incorporating the rejection in claim 5), Chen discloses the method further comprising: classifying the average trip count upon completion of each time interval subsequent to the introductory profiling phase to identify one or more loop transformation techniques applicable to the loop; and dynamically applying the one or more applicable loop transformation techniques to alter the code that relates to the loop if the trip count classification changes (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the beginning of program; for simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

14. **As to claim 7** (incorporating the rejection in claim 6), Chen discloses the method further comprising instrumenting the code relating to the loop with one or more counters to monitor the loop trip count (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 2nd Par. – in continuous profiling and optimization, program instrumentation is used to continuously collect profile information; procedures are selected and optimized based on instrumented profile information, and optimized procedures are hot-swapped back into the running program).

15. **As to claim 8** (incorporating the rejection in claim 7), Roediger discloses the method further comprising: counting consecutive intervals of time that do not have a trip count classification change; halting the trip count data collection if the number of consecutive intervals exceeds a threshold value; and removing the one or more monitoring counters from the code relating to the loop (Col. 9, Lines 25-51 – The Kth entry, where K<N, counts how many times the loop iterated exactly K times before exiting; The Nth entry counts how many times the loop iterated N or more times before exiting).

16. **As to claim 12** (incorporating the rejection in claim 1), Roediger discloses the method wherein collecting a loop trip count further comprises: collecting a trip count for the loop each time the loop is entered; and calculating an average trip count using a sequential plurality of collected trip counts over a determined number of iterations through the loop (Fig. 5; Col. 5, Line 46 through Col. 6, Line 7 – a known method in the prior art for determining the average number of executions per loop entry is shown as method 500 in Fig. 5; the total number of loop execution is then divided by the total number of loop entries to derive the average number of loop executions per loop entry).

17. **As to claim 13** (incorporating the rejection in claim 12), Roediger does not disclose the number of iterations through the loop is 50,000.

However, it is well known in the art of arbitrarily selecting a specific number of iterations through the loop in order to obtain the benefits know in the art.

18. **As to claim 16** (incorporating the rejection in claim 15), Roediger discloses the method wherein categorizing the loop trip count further comprises: determining a low trip count threshold value and a high trip count threshold value; classifying the trip count as being in a first condition if the trip count is equal to or below the low trip count threshold value; classifying the trip count as being in a second condition if the trip count is above the low trip count threshold value and below the high trip count threshold value; and classifying the trip count as being in a third condition if the trip count is equal to or above the high trip count threshold value (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

19. **As to claim 17** (incorporating the rejection in claim 16), Roediger does not explicitly disclose the method further comprising classifying the average trip count upon completion of each time interval subsequent to the introductory profiling phase to identify one or more loop transformation techniques applicable to the loop; and dynamically applying the one or more applicable loop transformation techniques to alter the code that relates to the loop if the trip count classification changes.

However, in an analogous art of dynamic trace selection using performance monitoring hardware sampling, Chen discloses the method further comprising classifying the average trip count upon completion of each time interval subsequent to the introductory profiling phase to identify one or more loop transformation techniques applicable to the loop; and dynamically applying the one or more applicable loop transformation techniques to alter the code that relates to the loop if the trip count classification changes (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the beginning of program; for simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Chen into the Roediger's system to further provide the method further comprising classifying the average trip count upon completion of each time interval subsequent to the introductory profiling phase to identify one or more loop transformation techniques applicable to the loop; and dynamically applying the one or more applicable loop transformation techniques to alter the code that relates to the loop if the trip count classification changes in Roediger system.

The motivation is that it would further enhance the Roediger's system by taking, advancing and/or incorporating Chen's system which offers significant advantages for capturing 58% of execution time across various SPEC2000 integer benchmarks using our profile and patching techniques on a relatively small number of frequently executed execution paths as once suggested by Chen (e.g., Abstract, 3rd Par.).

20. **As to claim 18** (incorporating the rejection in claim 16), Roediger discloses the method wherein the criteria further comprises whether the first condition is met (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

21. **As to claim 19** (incorporating the rejection in claim 16), Roediger discloses the method wherein the criteria further comprises whether the second condition is met (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

22. **As to claim 20** (incorporating the rejection in claim 16), Roediger discloses the method wherein the criteria further comprises whether the third condition is met (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-

56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

23. **As to claim 21** (incorporating the rejection in claim 15), Roediger discloses the method further comprising: counting consecutive intervals of time that do not have a trip count classification change; and halting the trip count data collection if the number of consecutive intervals exceeds a threshold value (Col. 9, Lines 25-51 – The Kth entry, where K<N, counts how many times the loop iterated exactly K times before exiting; The Nth entry counts how many times the loop iterated N or more times before exiting).

24. **As to claim 22** (incorporating the rejection in claim 21), Roediger discloses the method wherein the criteria further comprises whether the threshold value has been exceeded (Col. 9, Lines 25-51 – The Kth entry, where K<N, counts how many times the loop iterated exactly K times before exiting; The Nth entry counts how many times the loop iterated N or more times before exiting).

25. **As to claim 24** (incorporating the rejection in claim 23), Roediger discloses the machine readable medium wherein collecting a loop trip count further comprises: collecting a trip count for the loop each time the loop is entered; and calculating an average trip count using a sequential plurality of collected trip counts over an interval of time (Fig. 5; Col. 5, Line 46 through Col. 6, Line 7 – a known method in the prior art for determining the average number of executions per loop entry is shown as method 500

in Fig. 5; the total number of loop execution is then divided by the total number of loop entries to derive the average number of loop executions per loop entry).

26. **As to claim 25** (incorporating the rejection in claim 24), Roediger does not explicitly disclose the machine readable medium wherein the method further comprises executing the region of code for an introductory profiling phase time interval to establish an initial average trip count value.

However, in an analogous art of dynamic trace selection using performance monitoring hardware sampling, Chen discloses the machine readable medium wherein the method further comprises executing the region of code for an introductory profiling phase time interval to establish an initial average trip count value (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the beginning of program; for simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Chen into the Roediger's system to further provide the machine readable medium wherein the method further

comprises executing the region of code for an introductory profiling phase time interval to establish an initial average trip count value in Roediger system.

The motivation is that it would further enhance the Roediger's system by taking, advancing and/or incorporating Chen's system which offers significant advantages for capturing 58% of execution time across various SPEC2000 integer benchmarks using our profile and patching techniques on a relatively small number of frequently executed execution paths as once suggested by Chen (e.g., Abstract, 3rd Par.).

27. **As to claim 27** (incorporating the rejection in claim 25), Roediger discloses the machine readable medium wherein categorizing the trip count further comprises: determining a low trip count threshold value and a high trip count threshold value; classifying the trip count as being in a first condition if the trip count is equal to or below the low trip count threshold value; classifying the trip count as being in a second condition if the trip count is above the low trip count threshold value and below the high trip count threshold value; and classifying the trip count as being in a third condition if the trip count is equal to or above the high trip count threshold value (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

28. **As to claim 28** (incorporating the rejection in claim 27), Chen discloses the machine readable medium wherein the method further comprises: classifying the

average trip count upon completion of each time interval subsequent to the introductory profiling phase to identify one or more loop transformation techniques applicable to the loop; and dynamically applying the one or more applicable loop transformation techniques to alter the code that relates to the loop if the trip count classification changes (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the beginning of program; for simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

29. **As to claim 29** (incorporating the rejection in claim 28), Chen discloses the machine readable medium wherein the method further comprises instrumenting the code relating to the loop with one or more counters to monitor the loop trip count (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 2nd Par. – in continuous profiling and optimization, program instrumentation is used to continuously collect profile information; procedures are selected and optimized based on instrumented profile information, and optimized procedures are hot-swapped back into the running program).

30. **As to claim 30** (incorporating the rejection in claim 29), Roediger discloses the machine readable medium wherein the method further comprises counting consecutive intervals of time that do not have a trip count classification change; halting the trip count data collection if the number of consecutive intervals exceeds a threshold value; and removing the one or more monitoring counters from the code relating to the loop (Col. 9, Lines 25-51 – The Kth entry, where K<N, counts how many times the loop iterated exactly K times before exiting; The Nth entry counts how many times the loop iterated N or more times before exiting).

31. **As to claim 32** (incorporating the rejection in claim 31), Roediger discloses the system wherein the system: collects a trip count for the loop each time the loop is entered; and calculates an average trip count using a sequential plurality of collected trip counts over an interval of time (Fig. 5; Col. 5, Line 46 through Col. 6, Line 7 – a known method in the prior art for determining the average number of executions per loop entry is shown as method 500 in Fig. 5; the total number of loop execution is then divided by the total number of loop entries to derive the average number of loop executions per loop entry).

32. **As to claim 33** (incorporating the rejection in claim 32), Roediger does not explicitly disclose the system wherein the system executes the region of code for an introductory profiling phase time interval to establish an initial average trip count value.

However, in an analogous art of dynamic trace selection using performance monitoring hardware sampling, Chen discloses the system wherein the system executes the region of code for an introductory profiling phase time interval to establish an initial average trip count value (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the beginning of program; for simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Chen into the Roediger's system to further provide the system wherein the system executes the region of code for an introductory profiling phase time interval to establish an initial average trip count value in Roediger system.

The motivation is that it would further enhance the Roediger's system by taking, advancing and/or incorporating Chen's system which offers significant advantages for capturing 58% of execution time across various SPEC2000 integer benchmarks using our profile and patching techniques on a relatively small number of frequently executed execution paths as once suggested by Chen (e.g., Abstract, 3rd Par.).

33. **As to claim 35** (incorporating the rejection in claim 33), Roediger discloses the system wherein the system: determines a low trip count threshold value and a high trip count threshold value; classifies the trip count as being in a first condition if the trip count is equal to or below the low trip count threshold value; classifies the trip count as being in a second condition if the trip count is above the low trip count threshold value and below the high trip count threshold value; and classifies the trip count as being in a third condition if the trip count is equal to or above the high trip count threshold value (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

34. **As to claim 36** (incorporating the rejection in claim 35), Chen discloses The system wherein the system: classifies the average trip count upon completion of each time interval subsequent to the introductory profiling phase to identify one or more loop transformation techniques applicable to the loop; and dynamically applies the one or more applicable loop transformation techniques to alter the code that relates to the loop if the trip count classification changes (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 1st Par. – the original code is initially interpreted a few times before Dynamo generates code to directly execute; by limiting the number of times code is interpreted, frequently executed code is quickly moved into the code fragment cache, minimizing interpretation overhead; Sec. 3.7 – Trace Selection, 1st Par., 1st Bullet – in the first interval, a fixed number of samples are taken at the

beginning of program; for simple programs with one main loop, the first few seconds of profiling is often sufficient to capture all important traces).

35. **As to claim 37** (incorporating the rejection in claim 36), Chen discloses the system wherein the system instruments the code relating to the loop with one or more counters to monitor the loop trip count (Sec. 2.4 – Interpretation and Instrumentation Based Dynamic Optimization, 2nd Par. – in continuous profiling and optimization, program instrumentation is used to continuously collect profile information; procedures are selected and optimized based on instrumented profile information, and optimized procedures are hot-swapped back into the running program).

36. **As to claim 38** (incorporating the rejection in claim 37), Roediger discloses the system wherein the system: counts consecutive intervals of time that do not have a trip count classification change; halts the trip count data collection if the number of consecutive intervals exceeds a threshold value; and removes the one or more monitoring counters from the code relating to the loop (Col. 9, Lines 25-51 – The Kth entry, where K<N, counts how many times the loop iterated exactly K times before exiting; The Nth entry counts how many times the loop iterated N or more times before exiting).

37. Claims 4, 9-11, 26, and 34 are rejected under 35 U.S.C. 103(a) as being unpatentable over Roediger in view of Chen and further in view of Ghosh et al.,

(*Integrating High-Level Optimization in a Production Compiler: Design and Implementation Experience, April 2003, Springer-Verlag Berlin Heidelberg*) (hereinafter 'Ghosh')

38. **As to claim 4** (incorporating the rejection in claim 3), Roediger and Chen do not explicitly disclose the method wherein dynamically applying the one or more applicable code modification techniques further comprises applying one or more scalar transformation techniques to the loop upon receiving the initial average trip count value.

However, in an analogous art of integrating high-level optimization in a production compiler: design and implementation experience, Ghosh discloses the method wherein dynamically applying the one or more applicable code modification techniques further comprises applying one or more scalar transformation techniques to the loop upon receiving the initial average trip count value (Sec. 2 – Design Considerations Targeting the Itanium™ Processor, 1st Par., 3rd Bullet – maximize resource usage: scalar replacement of memory references, affine-condition un-switching, and load-pair insertion; Fig. 3 – Current phase-ordering o f optimizations in HLO, element of "Scalar Replacement"; P. 307, 1st Par. – loop fusion increases opportunities for reducing the overhead of array references by replacing them with references to compiler-generated scalar variables, 2nd Par., 4th Bullet – ability to expose ILP across loop back-edges has sufficiently higher benefit to tilt the balance towards expansion of scalar variables to enable loop distribution; Sec. 2.3 – Maximizing Resource Usage, 1st Par. – this category of optimizations includes scalar replacement of

memory references ...; Scalar replacement is a technique to replace memory references with compiler-generated temporary scalar variables that are eventually mapped to registers).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Ghosh into the Roediger-Chen's system to further provide the method wherein dynamically applying the one or more applicable code modification techniques further comprises applying one or more scalar transformation techniques to the loop upon receiving the initial average trip count value in Roediger-Chen system.

The motivation is that it would further enhance the Roediger-Chen's system by taking, advancing and/or incorporating Ghosh's system which offers significant advantages which in HLO (High-Level Optimizer), we have implemented numerous well-known and new transformations, and more importantly, we combined and extended these transformations in special ways so as to exploit the Itanium™ process architecture features for higher application performance as once suggested by Ghosh (e.g., Sec. 1 – Introduction, 3rd Par.).

39. **As to claim 9** (incorporating the rejection in claim 3), Roediger and Chen do not explicitly disclose the method further comprising: determining if the loop has a regular control flow graph and applying one or more scalar transformation techniques to the code relating to the loop and one or more loop transformations to the code relating to the loop upon receiving the initial trip count value if the control flow graph is regular.

However, in an analogous art of integrating high-level optimization in a production compiler: design and implementation experience, Ghosh discloses the method further comprising: determining if the loop has a regular control flow graph and applying one or more scalar transformation techniques to the code relating to the loop and one or more loop transformations to the code relating to the loop upon receiving the initial trip count value if the control flow graph is regular (Fig. 3 – current phase-ordering of optimizations in HLO, element of Scalar Replacement; Sec. 2.1 – Locality Optimizations, 1st Par., Lines 9-15 – they can also improve the effectiveness of other optimizations, such as scalar replacement..;)P.307, 1st Par. – Loop fusion increases opportunities for reducing the overhead of array references by replacing them with references to compiler-generated scalar variables; Sec. 2.3 – Maximizing Resource Usage, 1st Par. – scalar replace is a technique to replace memory references with compiler-generated temporary scalar variables that are eventually mapped to registers; scalar replacement, as implemented in Intel™ compiler for the Itanium™ processor, also replaces loop invariant memory references with scalar variables defined at appropriate levels of the loop nesting).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Ghosh into the Roediger-Chen's system to further provide the method further comprising: determining if the loop has a regular control flow graph and applying one or more scalar transformation techniques to the code relating to the loop and one or more loop transformations to the

code relating to the loop upon receiving the initial trip count value if the control flow graph is regular in Roediger-Chen system.

The motivation is that it would further enhance the Roediger-Chen's system by taking, advancing and/or incorporating Ghosh's system which offers significant advantages which in HLO (High-Level Optimizer), we have implemented numerous well-known and new transformations, and more importantly, we combined and extended these transformations in special ways so as to exploit the Itanium™ process architecture features for higher application performance as once suggested by Ghosh (e.g., Sec. 1 – Introduction, 3rd Par.).

40. **As to claim 10** (incorporating the rejection in claim 3), Roediger and Chen do not explicitly disclose the method further comprising: determining if the loop has substantial floating-point operations; and applying one or more scalar transformation techniques to the code relating to the loop and one or more loop transformations to the code relating to the loop upon receiving the initial trip count value if the loop has substantial floating-point operations.

However, in an analogous art of integrating high-level optimization in a production compiler: design and implementation experience, Ghosh discloses the method further comprising: determining if the loop has substantial floating-point operations; and applying one or more scalar transformation techniques to the code relating to the loop and one or more loop transformations to the code relating to the loop upon receiving the initial trip count value if the loop has substantial floating-point operations (Abstract, Lines 1-3 the High-Level Optimizer (HLO) is a key part of the

compiler technology that enable Itanium™ and Itanium™ 2 processor deliver leading floating-point performance at their introduction; P. 307, 1st Par., 1st Bullet – 128 floating-point and 128 general registers available in the Itanium processor family. This allows aggressive loop fusions without the risk of register pressure. The design allows for loop fusion across call boundaries, and code motion to enable loop fusion).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Ghosh into the Roediger-Chen's system to further provide the method further comprising: determining if the loop has substantial floating-point operations; and applying one or more scalar transformation techniques to the code relating to the loop and one or more loop transformations to the code relating to the loop upon receiving the initial trip count value if the loop has substantial floating-point operations in Roediger-Chen system.

The motivation is that it would further enhance the Roediger-Chen's system by taking, advancing and/or incorporating Ghosh's system which offers significant advantages which in HLO (High-Level Optimizer), we have implemented numerous well-known and new transformations, and more importantly, we combined and extended these transformations in special ways so as to exploit the Itanium™ process architecture features for higher application performance as once suggested by Ghosh (e.g., Sec. 1 – Introduction, 3rd Par.).

41. **As to claim 11** (incorporating the rejection in claim 6), Roediger discloses The method wherein applying loop transformations to the loop based on each trip count

classification further comprises applying loop peeling and loop unrolling transformations to the loop if the trip count classification is in the first condition; applying loop unrolling optimizations to the loop if the trip count classification is in the second condition (e.g., Fig. 11, steps 1130, 1140, 1150, 1160, steps 1132, 1142, 1152, 1162; Col. 8, Lines 10-56 – various examples are now presented to illustrate each of steps 1132, 1142, 1152, and 1162 in method 1100 of Fig. 11).

Roediger and Chen do not explicitly disclose software pipelining optimizations to the loop if the trip count classification is in the second condition; and applying software pipelining and data pre-fetching optimizations to the loop if the trip count classification is in the third condition.

However, in an analogous art of integrating high-level optimization in a production compiler: design and implementation experience, Ghosh discloses software pipelining (P. 311, Sub-Sec. of "Phase-ordering constraints", 2nd Par. – there is a handshake between pre-fetch and the software-pipe-liner that is part of the code-generator; as part of HLO (High-Level Optimizer), the compiler estimates the likelihood of a loop being pipelined; if a loop is predicted to be software-pipelined, an estimate of the initiation interval of the loop based on resource requirements is computed in advance;) optimizations to the loop if the trip count classification is in the second condition; and applying software pipelining and data pre-fetching (Fig. 5 – pre-fetch example illustrating the use of rotating registers; Sec. 2.4 – Data Pre-fetching, 1st Par. – data pre-fetching is an effective technique to hide memory access latency, 3rd Par. – the large number of registers available in the Itanium™ processor architecture enables pre-

fetch addresses to be stored in registers obviating the need for register spill and fill within loop) optimizations to the loop if the trip count classification is in the third condition.

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Ghosh into the Roediger-Chen's system to further provide software pipelining optimizations to the loop if the trip count classification is in the second condition; and applying software pipelining and data pre-fetching optimizations to the loop if the trip count classification is in the third condition in Roediger-Chen system.

The motivation is that it would further enhance the Roediger-Chen's system by taking, advancing and/or incorporating Ghosh's system which offers significant advantages which in HLO (High-Level Optimizer), we have implemented numerous well-known and new transformations, and more importantly, we combined and extended these transformations in special ways so as to exploit the Itanium™ process architecture features for higher application performance as once suggested by Ghosh (e.g., Sec. 1 – Introduction, 3rd Par.).

42. **As to claim 26** (incorporating the rejection in claim 25), Roediger and Chen do not explicitly disclose the machine readable medium wherein dynamically applying the one or more applicable code modification techniques further comprises applying one or more scalar transformation techniques to the loop upon receiving the initial average trip count value.

However, in an analogous art of integrating high-level optimization in a production compiler: design and implementation experience, Ghosh discloses the machine readable medium wherein dynamically applying the one or more applicable code modification techniques further comprises applying one or more scalar transformation techniques to the loop upon receiving the initial average trip count value (Sec. 2 – Design Considerations Targeting the Itanium™ Processor, 1st Par., 3rd Bullet – maximize resource usage: scalar replacement of memory references, affine-condition un-switching, and load-pair insertion; Fig. 3 – Current phase-ordering o f optimizations in HLO, element of "Scalar Replacement"; P. 307, 1st Par. – loop fusion increases opportunities for reducing the overhead of array references by replacing them with references to compiler-generated scalar variables, 2nd Par., 4th Bullet – ability to expose ILP across loop back-edges has sufficiently higher benefit to tilt the balance towards expansion of scalar variables to enable loop distribution; Sec. 2.3 – Maximizing Resource Usage, 1st Par. – this category of optimizations includes scalar replacement of memory references ...; Scalar replacement is a technique to replace memory references with compiler-generated temporary scalar variables that are eventually mapped to registers).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Ghosh into the Roediger-Chen's system to further provide the machine readable medium wherein dynamically applying the one or more applicable code modification techniques further comprises

applying one or more scalar transformation techniques to the loop upon receiving the initial average trip count value in Roediger-Chen system.

The motivation is that it would further enhance the Roediger-Chen's system by taking, advancing and/or incorporating Ghosh's system which offers significant advantages which in HLO, we have implemented numerous well-known and new transformations, and more importantly, we combined and extended these transformations in special ways so as to exploit the Itanium™ process architecture features for higher application performance as once suggested by Ghosh (e.g., Sec. 1 – Introduction, 3rd Par.).

43. **As to claim 34** (incorporating the rejection in claim 33), Roediger and Chen do not explicitly disclose the system wherein the system applies one or more scalar transformation techniques to the loop upon receiving the initial average trip count value.

However, in an analogous art of integrating high-level optimization in a production compiler: design and implementation experience, Ghosh discloses the system wherein the system applies one or more scalar transformation techniques to the loop upon receiving the initial average trip count value (Sec. 2 – Design Considerations Targeting the Itanium™ Processor, 1st Par., 3rd Bullet – maximize resource usage: scalar replacement of memory references, affine-condition un-switching, and load-pair insertion; Fig. 3 – Current phase-ordering o f optimizations in HLO, element of "Scalar Replacement"; P. 307, 1st Par. – loop fusion increases opportunities for reducing the overhead of array references by replacing them with references to compiler-generated scalar variables, 2nd Par., 4th Bullet – ability to expose ILP across loop back-edges has

sufficiently higher benefit to tilt the balance towards expansion of scalar variables to enable loop distribution; Sec. 2.3 – Maximizing Resource Usage, 1st Par. – this category of optimizations includes scalar replacement of memory references ...; Scalar replacement is a technique to replace memory references with compiler-generated temporary scalar variables that are eventually mapped to registers).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Ghosh into the Roediger-Chen's system to further provide the system wherein the system applies one or more scalar transformation techniques to the loop upon receiving the initial average trip count value in Roediger-Chen system.

The motivation is that it would further enhance the Roediger-Chen's system by taking, advancing and/or incorporating Ghosh's system which offers significant advantages which in HLO, we have implemented numerous well-known and new transformations, and more importantly, we combined and extended these transformations in special ways so as to exploit the Itanium™ process architecture features for higher application performance as once suggested by Ghosh (e.g., Sec. 1 – Introduction, 3rd Par.).

Conclusion

44. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

- Wu et al., Continuous Trip Count Profiling for Loop Optimizations in Two-phase Dynamic Binary Translators, Feb. 15, 2004, IEEE, pp. 1-10
- Wu et al., The Accuracy of Initial Prediction in Two-Phase Dynamic Binary Translators, Mar. 20, 2004, IEEE, pp. 1-12
- Lu et al., The Performance of Runtime Data Cache Prefetching in a Dynamic Optimization System, Dec. 3, 2003, IEEE, pp. 1-11

45. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW

June 5, 2007